Dieter Thomas

Schnellere Basic-Programme

Basic-Programme sind langsam. Zum Glück ist das für die meisten Anwendungen unerheblich. Das ändert sich, wenn z. B. umfangreiche Berechnungen durchzuführen sind, wo oft einige Programmteile bis zu mehrere hundert Male durchlaufen werden. Genau dort liegt der Ansatzpunkt, solche Programme schneller zu machen!

Die im folgenden genannten Beispiele und Daten beziehen sich auf den TRS-80. Die Ergebnisse sind daher nicht direkt auf andere Computer übertragbar, lassen sich aber leicht durch eigene Versuche nutzbringend anwenden.

FOR-NEXT lohnt sich

Die Motivation, Basic-Programme zu analysieren und zu beschleunigen, wird deutlich, wenn man die kleinen Programme in *Tabelle 1* betrachtet. Beim ersten Hinsehen könnte man vermuten, daß sie praktisch identisch sind: beide bilden eine Zählschleife bis 10 000 und melden sich dann mit „FERTIG". Der große Unterschied liegt in der Ausführungszeit! Programm A benötigt 22 Sekunden, Programm B hingegen 178 Sekunden, das ist über achtmal so lange. Dieses Beispiel zeigt deutlich, daß erhebliche Laufzeitreduzierungen möglich sind, und daß es sich lohnt, einzelne Basic-Befehle einmal näher zu betrachten.

Zeitmessung in Programmschleifen

Dazu wird die FOR-NEXT-Schleife in *Tabelle 2* benutzt. In Zeile 20 kann nun ein Basic-Befehl eingesetzt und seine Ausführungszeit bestimmt werden. Ohne Befehl in Zeile 20 dauert das Programm 22 Sekunden. Fügt man z.B. „20 Y=0" ein, wartet man 93 Sekunden auf die Fertigmeldung. Das heißt, die Ausführungszeit für einen einzelnen Befehl „Y=0" beträgt gerade (93–22)/10 000 s = 7,1 ms. Auf diese Weise lassen sich sehr einfach die Ausführungszeiten verschiedener Basicbefehle bestimmen. Hier sollen nur die Befehle behandelt werden, die oft in Schleifen vorkommen, da nur hier effektiv Zeit gespart werden kann. In *Tabelle 3* sind die verschiedenen denkbaren Zuweisungsbefehle und ihre jeweiligen Ausführungszeiten abgedruckt. Die mit Klammern versehenen Zuweisungen, z.B. „(B=1)", bedeuten, daß dieser Befehl außerhalb der Schleife steht, beispielsweise in Zeile 5.

LET kostet Speicherplatz, ist aber schneller

Wie man sieht, differieren die Zeiten mitunter ganz erheblich. So sind die Zeiten für Y=0 und Y=10 000 fast gleich, wogegen Y=0,001 zur Ausführung mehr als doppelt so lange braucht. Man erkennt, daß man Zuweisungen der Art Y=A(I) oder A(1)=5 vermeiden sollte, denn sie sind um ein Drittel langsamer als z.B. Y=5. Etwa 17 % Zeitgewinn erreicht man durch die Zuordnung zweier Variablen, etwa Y=B. Die wesentlichste Methode besteht in der Benutzung des LET-Befehls. So ist LET Y=B um rund 40 % schneller als Y=B!
In *Tabelle 4* findet man eine kleine Zusammenstellung arithmetischer Operationen. Die Zeiten für Addition und Subtraktion sind etwa gleich. Daß Multiplikation und Division länger dauern, ist nicht erstaunlich. Wichtig ist, daß sich wieder die gleichen Ergebnisse finden lassen. Rechnungen mit ganzzahligen Variablen und LET-Anweisung sind am schnellsten. In *Tabelle 5* wird es sehr deutlich, daß der TRS-80 beim Variablenrechnen schneller ist als beim Rechnen mit Zahlen. Man beachte, daß in diesen beiden Beispielen die Schleifen nur jeweils 1000 Mal durchlaufen werden. Der Unterschied ist frappierend! Programm C benötigt 9 Sekunden, Programm D 36 Sekunden. Das ist viermal solange! Für die im Level 1 enthaltenen Funktionen INT, ABS und RND gilt ebenfalls das bisher Gesagte: LET-Anweisung und möglichst ganzzahlige Variablen benutzen.

THEN kostet Zeit

Interessant sind neben arithmetischen Befehlen besonders auch IF/THEN-Abfragen, da sie sehr häufig in Rechenschleifen benutzt werden. Bei einer näheren Analyse muß vor allem auf den Unterschied aufmerksam gemacht werden, ob die Abfrage wahr oder falsch ist. Betrachtet man z.B. die folgende Abfrage (unter der Voraussetzung A=1):
IF A=1 THEN B=7
so werden dafür 14 Millisekunden benötigt. Ist jedoch A=2 und wird statt THEN die Anweisung LET eingesetzt, als:
IF A=1 LET B=7
dann beträgt die Ausführungszeit nur die Hälfte, nämlich nur 7 Millisekunden.

Zusammenfassung der Ergebnisse

Die gewonnenen Ergebnisse sollen noch einmal kurz aufgelistet und durch allgemeine Regeln ergänzt werden:

1. LET-Anweisung benutzen.
2. Häufig gebrauchte Variablen definieren (Beispiel Y=0,0567) und damit arbeiten.
3. Möglichst ganze Zahlen verwenden.
4. Schleifen mit FOR/NEXT und nicht durch Inkrementieren einer Variablen aufbauen.
5. Keine unnötigen Befehle innerhalb von Schleifen verwenden, auch
6. REM kostet Zeit.
7. IF/THEN-Abfragen sollten möglichst lange „falsch" sein.
8. Mehrere Befehlen in eine Zeile schreiben.

Diese Richtlinien sollten vor allem innerhalb von Schleifen befolgt werden, denn dort führen sie zu wesentlichen Laufzeitreduzierungen. Wie schon erwähnt, gelten diese Regeln streng nur für das Level-1-Basic des TRS-80. Für andere Rechner müssen die Ergebnisse überprüft werden.
Abschließend sollen noch zwei kurze Programme vorgestellt werden (*Tabelle*

6), die die Wirksamkeit der aufgestellten Regeln demonstrieren. Beide Programme berechnen die Volumina und Oberflächen von Kugeln mit den Radien R=1 bis R=200. Im ersten Programm werden die gefundenen Ergebnisse konsequent berücksichtigt; das zweite soll als Gegenbeispiel dienen. Die Laufzeiten sprechen für sich: 20 Sekunden einerseits, 36 Sekunden andererseits.

Tabelle 1: Programmschleife mit FOR-NEXT oder Variablen-Inkrementierung

FOR-NEXT-Version:

```
10 FOR X=1 TO 10 000
20 NEXT X
30 PRINT„FERTIG"
40 END
```

Inkrementierungs-Version:

```
10 X=1
20 X=X+1
30 IF X < 10 000 THEN 20
40 PRINT „FERTIG"
50 END
```

Tabelle 2: Messung der Dauer eines Basic-Befehls

```
10 FOR X=1 TO 10 000
20 ... BASICBEFEHL ...
30 NEXT X
40 PRINT „FERTIG"
50 END
```

Tabelle 3: Ausführungszeiten einiger Basic-Befehle

Befehl	Zeit/s
Y=0	0,0071
Y=10 000	0,0077
Y=0,001	0,0161
Y=B (B=1)	0,0059
A(1)=4	0,0107
A(I)=4 (I=1)	0,0096
y=A(1)	0,0097
LET Y=0	0,0046
LET Y=B (B=1)	0,0034
LET A (1)=4	0,0083

Tabelle 4: Zeitbedarf arithmetischer Operationen

Befehl	Ausführungszeit/s			
	+	–	*	/
Y=1 ? 1	0,0100	0,0100	0,0112	0,0118
Y=A ? B	0,0076	0,0078	0,0088	0,0092
LET Y=1 ? 1	0,0074	0,0074	0,0085	0,0094
LET Y=A ? B	0,0051	0,0054	0,0063	0,0068

Tabelle 5: Vergleich von Variablen- und Zahlenoperationen

Rechnen mit Variablen

```
 5 A=0,1234:B=0,5678
10 FOR X=1 TO 1000
20 LET Y=A*B
30 NEXT X
40 PRINT „FERTIG"
50 END
```

Rechnen mit Zahlen

```
10 FOR X=1 TO 1000
20 Y=0,1234*0,5678
30 NEXT X
40 PRINT „FERTIG"
50 END
```

Tabelle 6: Errechnen von Kugel-Oberfläche und Volumen

Schnelle Version

```
10 A=4*3.14159
20 FOR R=1 TO 200:LET V=A*R*R*R/3
30 LET F=A*R*R:PRINT V,F:NEXT R
40 PRINT „FERTIG"
50 END
```

Langsame Version

```
10 R=0
20 R=R+1
30 V=4*0,33333*3.14159*(R*R*R)
40 F=4*3.14159*(R*R)
50 PRINT V,F
60 IF R <=200 THEN 20
70 PRINT „FERTIG"
80 END
```

System-spezifisches

Die hier gezeigten Beispiele beziehen sich auf das Level-1-Basic des TRS-80, das eingegebene Programme als reine ASCII-Zeichenfolge im Arbeitsspeicher ablegt. Die Basic-Befehlsworte werden dabei nicht in Kurzbytes (Tokens) verschlüsselt – im Gegensatz zum Level-2-Basic des TRS-80 und zu den meisten anderen Basic-Interpretern. Trotzdem sind die hier gewonnenen Erkenntnisse im Prinzip auch für andere Computer anwendbar.

Mehr Leistung für Apple II

Orange Speed heißt ein neues Produkt, das den Apple II insbesondere für technisch-wissenschaftliche Anwendungen auf Vordermann bringt. Es besteht aus einer Einsteckkarte, die mit dem Arithmetik-Prozessor Am9511A ausgestattet ist, einem ROM-Adapter, zwei Disketten und einem umfangreichen Handbuch. Der Arithmetik-Prozessor kann von einer beliebigen Sprache aus angesprochen werden. Er erledigt Rechenoperationen bis zu 100mal schneller als der 6502. Neben Ganzzahlen (±32768 oder ±2147483647) verarbeitet er auch Gleitkommazahlen ($\pm 0{,}9223367 \cdot 10^{19}$). Die möglichen Rechenoperationen reichen von den vier Grundrechenarten bis zum Wurzelziehen, zu trigonometrischen und Exponentialfunktionen. Daneben sind zahlreiche Befehle für Stackmanipulationen und Formatumwandlungen vorgesehen.

Der eigentliche Clou von Orange Speed aber ist die Software. Sie besteht im wesentlichen aus einem Compiler für „Metalanguage" – einer stackorientierten Sprache, die nach Art mancher Taschenrechner in der umgekehrten polnischen Notation arbeitet. Diese Sprache kennt als einziges Element den Befehl. Ausgehend von einem Grundbefehlssatz „baut" sich der Programmierer eigene Befehle, die dem Wortschatz angefügt werden. Ein Programm besteht schließlich aus einem einzigen Befehl, der die gestellte Aufgabe ausführt. Nachteil: Der Programmierer ist für die Verwaltung des Stack zu jeder Zeit selbst verantwortlich und wird vom System kaum auf Fehler aufmerksam gemacht. Leute, die es gewöhnt sind, Basic-Programme zu erstellen, dürften sich anfangs damit sehr schwer tun. Dies sind allerdings nicht die typischen Anwender. Besonders geeignet ist die Sprache zusammen mit dem Zusatzprozessor für Grafikanwendungen, bei denen viel berechnet werden muß. Wer schon einmal am Bildschirm erlebt hat, wie mühsam Punkt für Punkt an eine Kurve angefügt wird, der weiß den erheblichen Geschwindigkeitszuwachs zu schätzen. Befehle zum Erstellen von Polygonzügen und zum Einfügen von Text in die Grafikseite erhöhen den Komfort.

Das ganze System ist kompatibel zu DOS 3.2/3.3, Applesoft, CP/M und Pascal. Hervorheben muß man die ausgezeichnete deutsche Dokumentation.

Rudolf Hofer